14. ⋆En este ejercicio diseñamos un algoritmo para encontrar ciclos en un digrafo. Decimos que un digrafo es *acíclico* cuando no tiene ciclos dirigidos. Recordar que un (di)grafo es *trivial* cuando tiene un sólo vértice.

   a) Demostrar con un argumento constructivo que si todos los vértices de un digrafo D tienen grado de salida mayor a 0, entonces D tiene un ciclo.

   b) Diseñar un algoritmo que permita encontrar un ciclo en un digrafo D cuyos vértices tengan todos grado de salida mayor a 0.

   c) Explicar detalladamente (sin usar código) cómo se implementa el algoritmo del inciso anterior. El algoritmo resultante tiene que tener complejidad temporal $O(n+m)$.

   d) Demostrar que un digrafo D es acíclico si y solo si D es trivial o D tiene un vértice con $d_{out}(v) = 0$ tal que $D \setminus \{v\}$ es acíclico.

   e) A partir del inciso anterior, diseñar un algoritmo que permita determinar si un grafo D tiene ciclos. En caso negativo, el algoritmo debe retornar una lista $v_1, \ldots, v_n$ de vértices tales que $d_{out}(v_i) = 0$ en $D \setminus \{v_1, \ldots, v_{i-1}\}$ para todo i. En caso afirmativo, el algoritmo debe retornar un ciclo.

   f) Explicar detalladamente (sin usar código) cómo se implementa el algoritmo del inciso anterior. El algoritmo resultante tiene que tener complejidad temporal $O(n+m)$.

Sea G un digrafo tq $d_{out}(v) > 0 \ \forall v \in G_V$.

Lema 1: Existe un recorrido R arbitrariamente largo en G.
Demo: Sea $v \in G_V$ el comienzo del recorrido. Como $d_{out}(v) > 0$, existe una arista $e \in G_E$, $e = (v, u)$ para algún $u \in G_V$. Agregamos u al recorrido. Como también vale $d_{out}(u) > 0$, con el mismo argumento podemos agregar otro vértice más al recorrido. En general, como todos los vértices de G tienen $d_{out} > 0$, siempre se puede extender el recorrido generando así un recorrido R de cualquier largo.

Sea R un recorrido tq $|R| = n+1$, que existe en G por lema 1. Como $|G_V| = n$ (hay solo n vértices), por el principio del palomar el recorrido R visita algún vértice v al menos 2 veces. Por lo tanto hay un ciclo en R, y consecuentemente hay uno en G.

```
next[1...n] ← ⊥                  O(n)
v ← 1
while next[v] = ⊥ :              O(n)
    u ← N(v)[0]                  existe porque d_out(v) > 0 ∀v ∈ G_V
    next[v] = u
    v ← u
C ← [v]
r ← v
do:
    v ← next[v]
    push(C, v)
while r ≠ v                      O(n)
return C
```

1) Inicializamos un vector para trackear hacia dónde vamos desde cada vértice.
2) Comenzamos el recorrido en el primer vértice ($v=1$).
3) En cada iteración elegimos algún vecino u de v.
El algoritmo toma el primero de la lista de adyacencia, pero eso es solo un detalle de implementación.
4) Marcamos a u como el siguiente de v, y repetimos el proceso hasta llegar a un vértice que ya tiene asignado un siguiente. Ahí encontramos el ciclo.
5) Reconstruimos el ciclo en el vector C recorriendo las aristas codificadas en next. Cuando volvimos a r dimos 1 vuelta por el ciclo.

Digrafo G es acíclico $\iff$ G es trivial ó

$\exists v \in G_V$, $d_{out}(v) = 0$ tq $G - v$ es acíclico

$\Leftarrow$

Si G es trivial entonces $|G_V| = 1$. Un ciclo requiere al menos 2 vértices, por lo tanto G trivial es acíclico.

Veamos el otro caso. Si existe un vértice v en G con $d_{out}(v) = 0$ tq $G - v$ es acíclico, entonces G también es acíclico porque el vértice v no puede ser parte de ningún ciclo si no hay aristas que vayan desde v hacia algún otro vértice de G.

$\Rightarrow$

Si G es acíclico entonces una posibilidad es que G sea trivial ya que el digrafo trivial no tiene ciclos.

Suponiendo que G no es trivial, en el inciso a) demostramos que:

$\forall v \in G_V$, $d_{out}(v) > 0 \Rightarrow G$ tiene un ciclo

Utilizando el contrarrecíproco vale que:

G no tiene ciclos $\Rightarrow \exists v \in G_V$, $d_{out}(v) = 0$

Veamos que $G - v$ es acíclico. Vale trivialmente porque ya estamos suponiendo que G es acíclico, entonces al sacarle un vértice va a seguir siendo acíclico.

$\boxtimes$

1) Inicializamos un vector $D[1 \ldots n]$ con $D[i] = d_{out}(v_i)$ $O(n+m)$

2) Inicializamos un stack S vacío

3) Pusheamos en S todos los vértices con $d_{out} = 0$ $O(n)$

4) Mientras $|S| > 0$: $O(n)$

    Popeamos v de S

    Para cada vecino u que tenga una arista (u, v): $\Sigma = O(m)$

        $D[u] \leftarrow D[u] - 1$

        Si $D[u] = 0$: pusheamos u en S

5) Construimos un conjunto R de vértices a partir de D: $O(n)$

    $v \in R \iff D[v] > 0$

6) Si $|R| = 0$:

    Retornamos que no hay ciclos

7) Si no, utilizamos el algoritmo del inciso b) para retornar el ciclo en R. $O(n+m)$